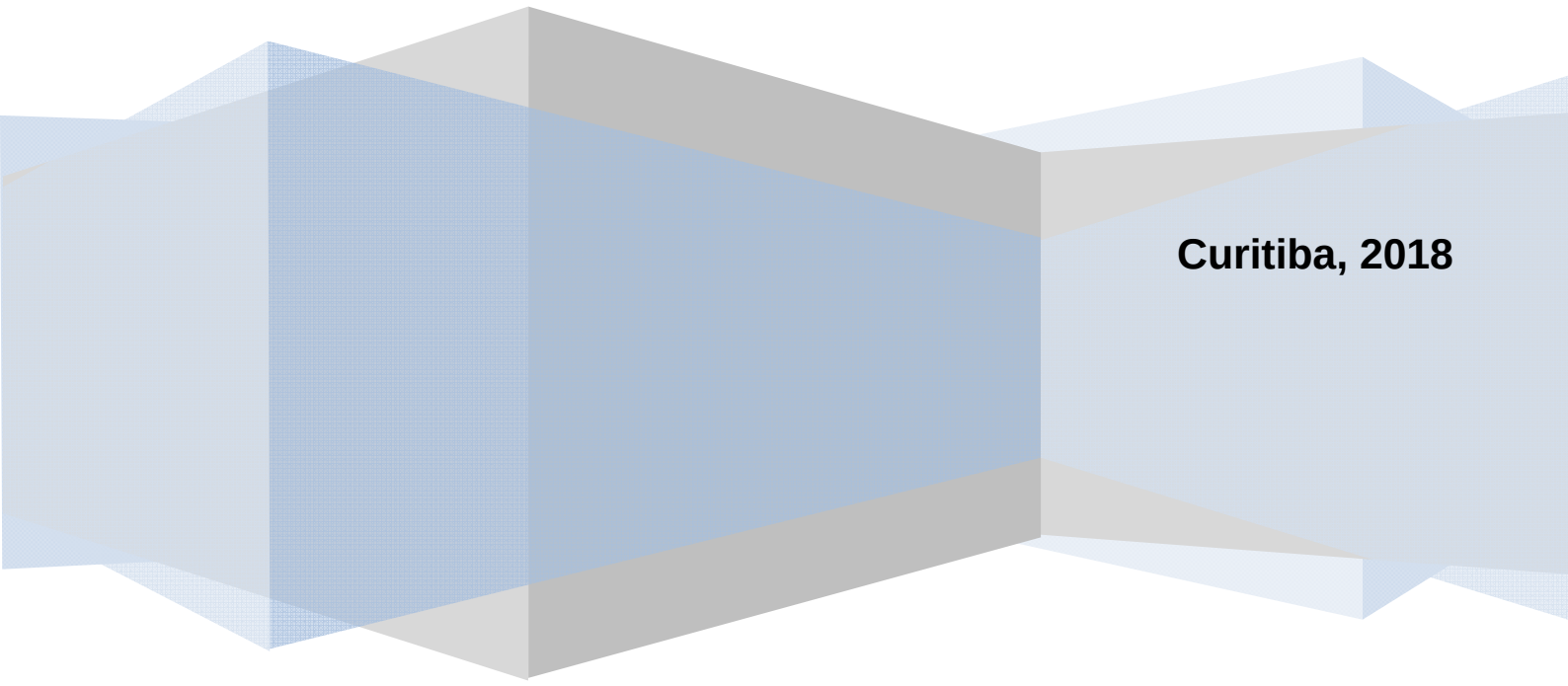
Introdução à linguagem de programação Delphi

Cicero Aparecido Bezerra



Curitiba, 2018

SUMÁRIO

1 SOBRE O AUTOR

Possui graduação em Informática pela Universidade do Vale do Rio dos Sinos (1992), mestrado em Engenharia de Produção pela Universidade Federal de Santa Catarina (2001), doutorado em Engenharia de Produção pela Universidade Federal de Santa Catarina (2007) e estágio pós-doutoral em Gestão Estratégica da Informação e do Conhecimento pela Pontifícia Universidade Católica do Paraná (2012). Atualmente é professor Associado nível II da Universidade Federal do Paraná. Tem experiência em profissional em desenvolvimento, implantação e gestão de Sistemas de Informação e, Processos de Produção. Enquanto docente, leciona disciplinas alinhadas ao desenvolvimento de Sistemas de Informação e Análise de Dados. Como pesquisador, tem voltado sua atenção aos Métodos e Técnicas de Análise de Dados, aplicados à Gestão do Conhecimento e Inovação.

2 APRESENTAÇÃO

O material ao qual vocês estão tendo acesso apresenta, passo a passo, uma maneira para o desenvolvimento de programas computacionais na linguagem Delphi. Foi desenvolvido com a expectativa que sua leitura sequencial conduza, naturalmente, à construção de programas de computador, a partir de uma linguagem coloquial, próxima ao leitor.

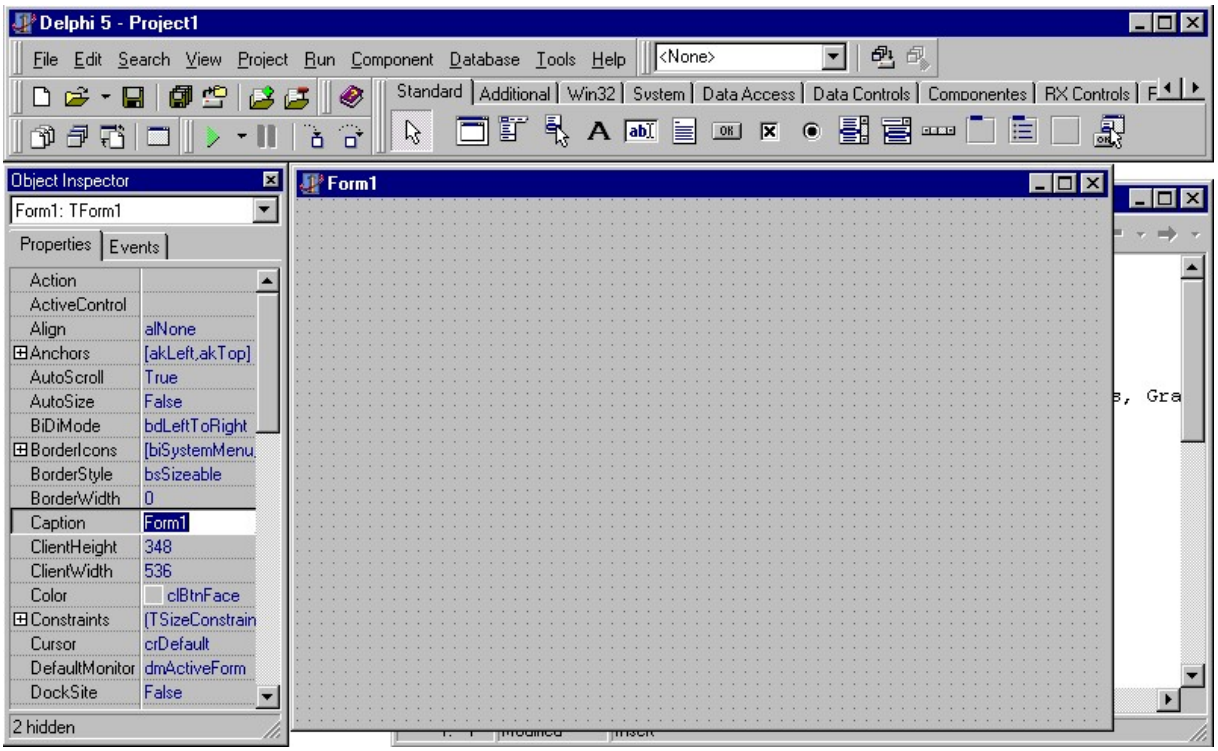
A primeira parte consiste em uma visão geral do ambiente de programação Delphi. A seguir são apresentadas as instruções e operações básicas requeridas na linguagem Delphi. Logo depois, as estruturas de seleção simples, composta e de múltipla escolha são abordadas. Na sequência, é a vez das estruturas de repetição, com teste de início e no fim e, com variável de controle. A próxima seção traz algumas dicas para uma melhor codificação, para logo em seguida apresentar as estruturas que permitem a manipulação de dados. Finalmente, será abordado o armazenamento de informações.

Espero que este material possa ser útil na compreensão básica dos conceitos e comandos necessários à programação de computadores em Delphi.

Bons estudos...

3 AMBIENTE DE DESENVOLVIMENTO DELPHI

O ambiente de programação Delphi reúne os recursos de um compilador, editor de códigos e editor de componentes. É dividido, basicamente, em quatro janelas.

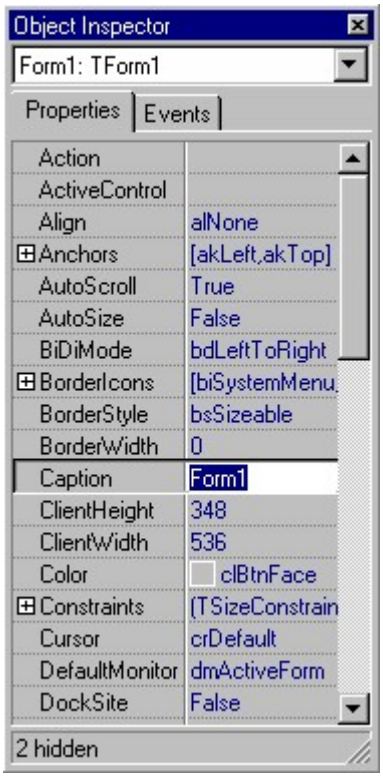


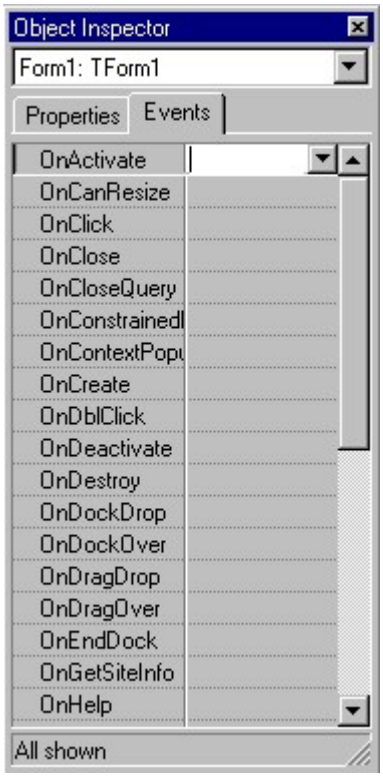
Janela principal

É a responsável pelo controle geral do projeto que está sendo desenvolvido. É composta por uma barra de menus, barra de ferramentas e paleta de componentes.

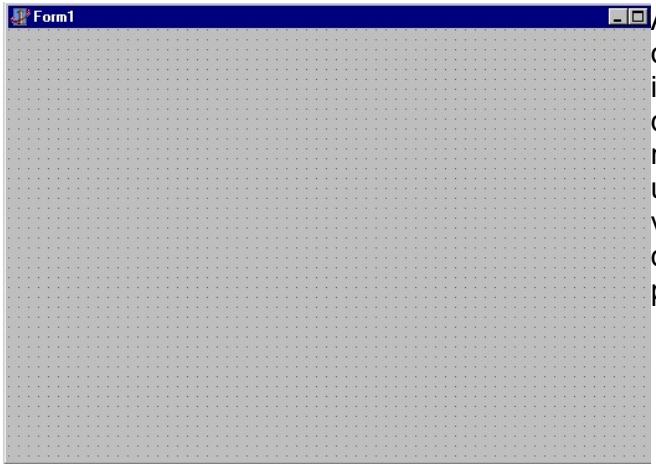
Object inspector

Apresenta um botão do tipo drop-down que permite a seleção de algum componente disponível. Logo ao lado existem duas guias: **proprerties** (representando as características de um determinado componente, sendo que as características podem ser alteradas automaticamente durante a execução do programa) e **events** (representando o comportamento de determinado componente, através do código inserido em cada evento). É responsável por gerenciar toda a parte relativa aos componentes e suas ações.





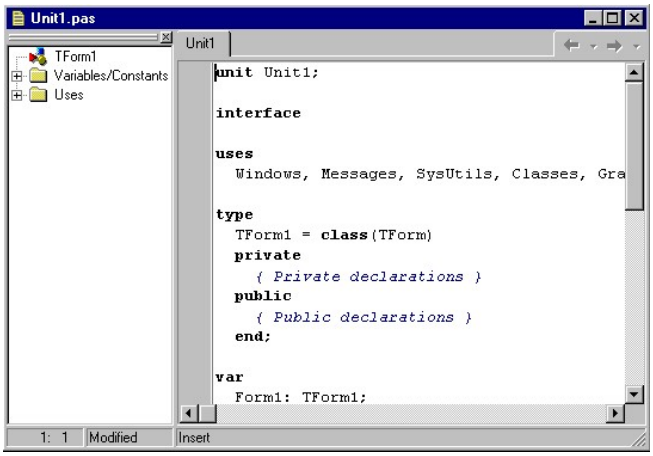
Form1



A janela Form1 apresenta o local onde os componentes são inseridos e interagem entre si, gerenciando uma determinada rotina de programa ou até mesmo um sistema. É a parte visual de uma aplicação em Delphi, onde o que você distribui no Form será próximo daquilo que será executado quando o programa estiver em funcionamento.

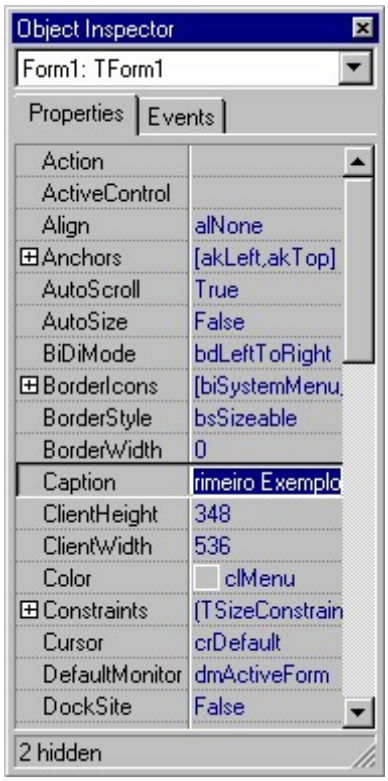
Unit1.pas

A janela Unit1.pas fica posicionada logo abaixo do Form1 e é o local o código-fonte está sendo montado. Ao implementar componentes no Form1, o Delphi automaticamente implementa as linhas de código. Apesar deste recurso, inevitavelmente teremos que intervir diretamente no código, para que o mesmo se adapte corretamente à necessidade da aplicação em desenvolvimento.



4 PROGRAMANDO EM DELPHI

Alterando propriedades de componentes



Antes de iniciarmos é importante esclarecer que devido ao número elevado de propriedades para cada tipo de componente, iremos estudá-las à medida em que formos introduzindo novos componentes com propriedades próprias e que, infelizmente, nem todas as propriedades serão vistas.

Nosso primeiro componente é o formulário (**Form1**). Vamos alterar seu nome diretamente no **Object Inspector**. Para isto, basta acessar a propriedade **Caption** e alterá-la para “Primeiro Exemplo”.

Para alteramos a cor basta escolher a propriedade **Color** e selecionar a cor desejada.

Para alterarmos a posição onde a janela será exibida deve-se selecionar a propriedade **Align** e escolher a posição desejada.

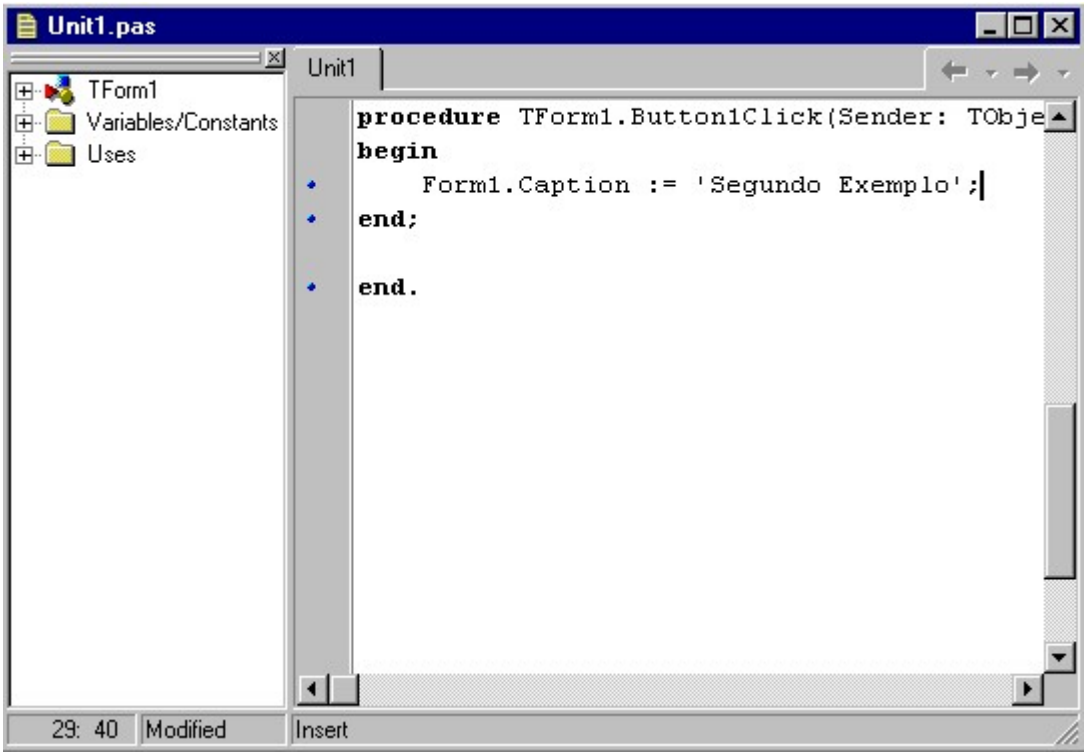
Executando e encerrando um programa

Para visualizarmos o resultado do programa basta clicarmos no botão ► ou menu **Run**, comando **Run**, ou ainda pressionar a tecla **F9**. Para interrompermos a execução do programa, basta clicarmos no botão **Fechar** no canto superior direito do formulário.

Introduzindo novos componentes

Vamos inserir dois componentes do tipo **Button**, localizados na paleta de componentes. Para isto basta clicar no componente e clicar no local desejado para o mesmo no formulário. No **Button1** iremos mudar seu **Caption** para Título e o **Button2** para Cor, clicando no componente instalado no formulário e acessando sua propriedade **Caption** no **Object Inspector**.

A partir daí, nosso segundo programa será fazer com que o título do formulário mude para “Segundo Programa” ao clicarmos o botão **Título** e se clicarmos no botão **Cor**, a cor da tela mudará para vermelho, por exemplo. Para isto basta dar um duplo clique no botão **Título**. Irá abrir a **Unit1** e iremos atribuir “Segundo Exemplo” para o componente **Form1**, propriedade **Caption**.



Vamos voltar ao formulário (basta clicar na sua barra de título ou pressionar a tecla **F12**, ou ainda, menu **View**, comando **Toggle Form/Unit**) e efetuar o mesmo procedimento para o botão **Cor**, alterando a propriedade **Color** para clRed.

Salvando o projeto

Antes de salvar o programa é importante saber que o Delphi gera automaticamente vários arquivos e, sendo assim, é aconselhável criar uma pasta para cada projeto. Entre os arquivos criados, podemos citar:

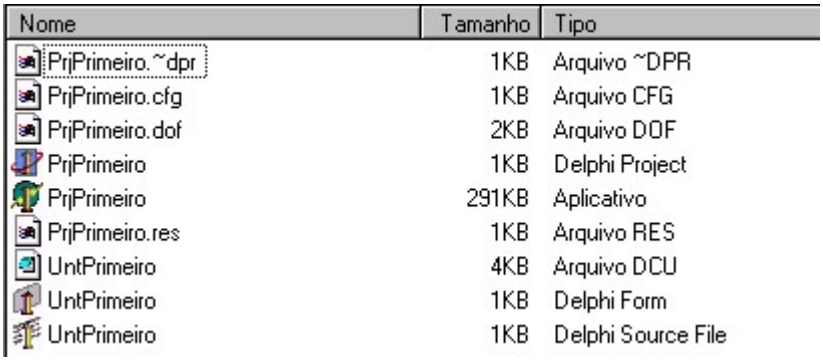
Tipo de Arquivo	Descrição
Project (.DPR)	Armazena informações sobre form's e unit's
Unit (.PAS)	Armazena o código-fonte
Form (.DFM)	Armazena informações sobre o formulário (possui associação com o .PAS)
Resource (.RES)	Armazena recursos utilizados no projeto (ícones, bitmaps, etc)
Compiled Unit (.DCU)	É o arquivo resultante da compilação de algum arquivo de código (.PAS)

Para salvar, basta acessar o menu **File**, comando **Save All** ou clicar no botão correspondente na barra de ferramentas.

O primeiro elemento a ser salvo será a **Unit** (arquivo do código-fonte). Para que não aja confusão depois do projeto salvo, sugere-se salvá-la com as três primeiras letras indicando tratar-se de uma **Unit**. Exemplo: UntPrimeiro.

O próximo elemento será o **Project** (arquivos utilizados no desenvolvimento do programa). Como não podem haver nomes iguais entre Unit's e Form's sugere-se salvá-lo com as três primeiras letras indicando tratar-se de um **Project**. Exemplo: PrjPrimeiro.

Para acessar o **Project** deve-se clicar no arquivo PrjPrimeiro do tipo **Delphi Project**. Para executar o programa deve-se clicar no arquivo PrjPrimeiro do tipo **Aplicativo**.



Nome	Tamanho	Tipo
PrjPrimeiro.~dpr	1KB	Arquivo ~DPR
PrjPrimeiro.cfg	1KB	Arquivo CFG
PrjPrimeiro.dof	2KB	Arquivo DOF
PrjPrimeiro	1KB	Delphi Project
PrjPrimeiro	291KB	Aplicativo
PrjPrimeiro.res	1KB	Arquivo RES
UntPrimeiro	4KB	Arquivo DCU
UntPrimeiro	1KB	Delphi Form
UntPrimeiro	1KB	Delphi Source File

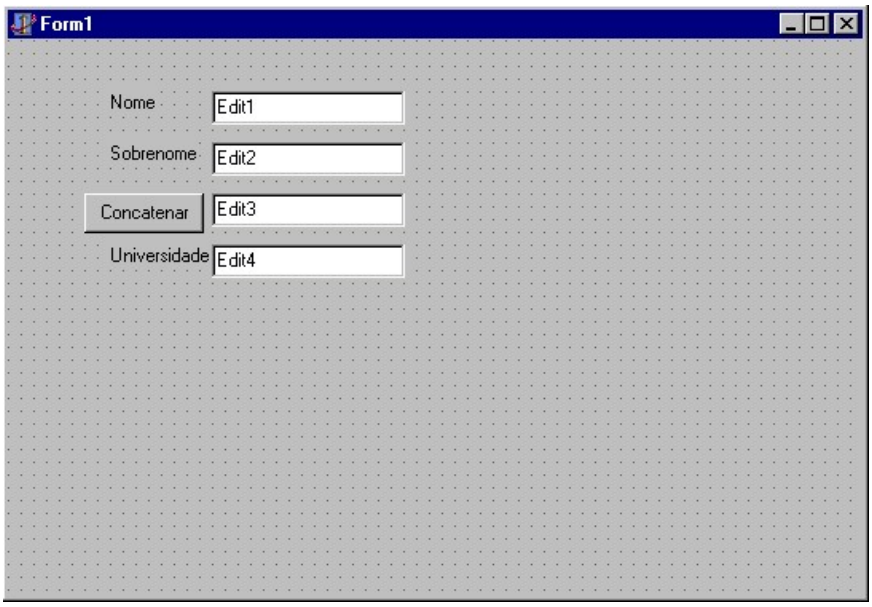
Trabalhando com variáveis

Vamos abrir um novo **Project** acessando o menu **File**, comando **New Application** ou clicando no botão **New** e selecionando **Application** na paleta **New**. O segundo programa irá permitir que se digite o nome e o sobrenome separadamente (em caixas de texto – **Edit** – distintas) e, após clicar no botão Concatenar irá mostrar nome e sobrenome juntos; além disso ao clicarmos em uma nova caixa de texto (**Edit**), a mesma irá mostrar o nome da universidade onde estudamos.

O primeiro passo é clicar, na paleta de componentes, no componente **Label** e clicarmos no **Form**, na posição desejada para nomearmos um campo a ser digitado. Na propriedade **Caption** deste componente iremos alterar para Nome. Repetiremos a operação com novo **Label**, porém com o **Caption** alterado para Sobrenome.

A seguir iremos clicar no componente **Button** e posicioná-lo logo abaixo do **Label** Sobrenome, alterando o **Caption** do **Button** para Concatenar. Abaixo do **Button** Concatenar iremos posicionar novo **Label** com o **Caption** Universidade.

O terceiro passo é clicar no componente **Edit**, posicionando-o no **Form**, para que o mesmo possa receber e mostrar valores. Portanto, neste caso, teremos quatro **Edit's**. A aparência do **Form** será semelhante à figura a seguir:



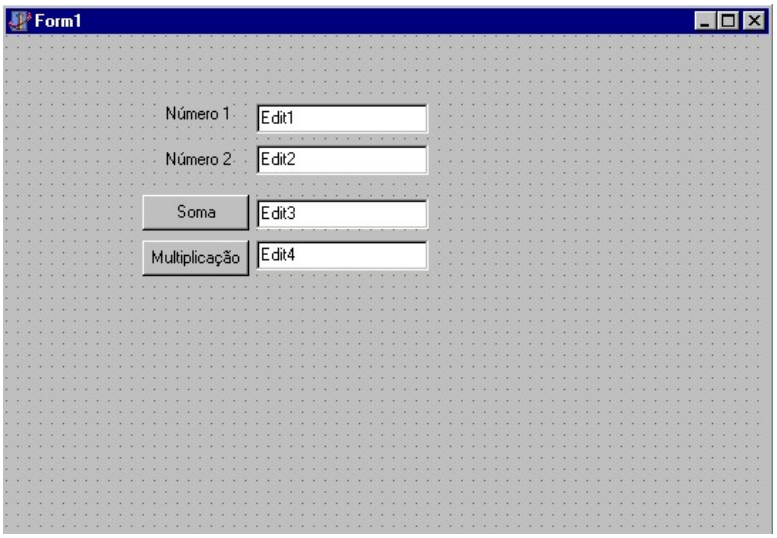
No **Button** Concatenar iremos escrever uma linha de código para permitir que ao clicarmos sobre ele, mostre no objeto **Edit3**, o nome (digitado no objeto **Edit1**) e o sobrenome (digitado no objeto **Edit2**) concatenados. Para isto devemos atribuir à propriedade **Text** do objeto **Edit3**, o conteúdo digitado na propriedade **Text** do objeto **Edit1** e do objeto **Edit2**.

Para apresentarmos a constante 'UNIPAR' na propriedade **Text** do objeto **Edit4**, ao clicarmos sobre ele, devemos selecioná-lo, acessar a guia **Events** do **Objetc Inspector**, clicar duas vezes no evento **OnClick**. Após isto digitar o código onde a propriedade **Text** do objeto **Edit4** receba a constante 'UNIPAR'. Feito isto, basta executar o programa.

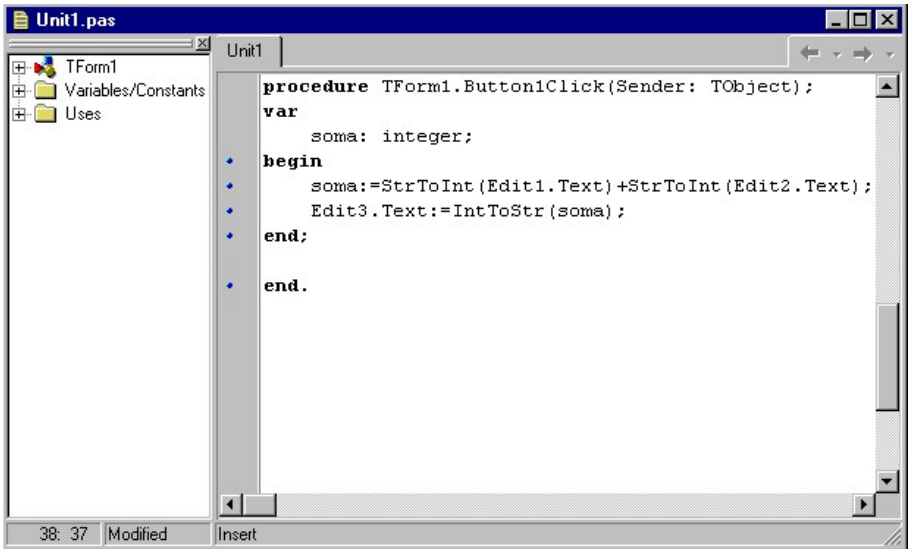
Com este exemplo é possível deduzir que não existe padrão para escolhas de eventos ou propriedades, dependendo unicamente do problema a ser resolvido.

O próximo programa irá trabalhar com variáveis numéricas, onde iremos digitar dois números inteiros em **Edit's** distintos e efetuar sua soma e sua multiplicação.

O primeiro passo é inserir os componentes **Label** (para indicar o número 1 e o número 2), logo em seguida inserir os componentes **Edit** (para receber o número 1 e número 2 e mostrar o resultado da soma e da multiplicação), finalmente inserir dois **Buttons** (para calcular a soma e a multiplicação). O resultado será semelhante ao ilustrado na figura abaixo:



O próximo passo é digitar o código referente à soma no **Button** Soma, clicando duas vezes neste componente. Como os **Edit's** são do tipo **Text** (armazenam **string's**), temos que criar uma variável que receba a soma dos valores digitados em **Edit1** e **Edit2** convertidos em inteiro (função **StrToInt**). Após isto, alterar este valor inteiro para string (**IntToStr**) para podermos atribuí-lo ao **Edit3**. A **Unit1.pas** assumirá a seguinte aparência:



O mesmo procedimento deve ser utilizado para efetuar a multiplicação. Outra função de conversão bastante empregada é aquela que transforma string em número fracionário (**StrToFloat**) e número fracionário em string (**FloatToStr**).

EXERCÍCIOS

1. Faça um programa em Delphi para determinar e mostrar o valor das seguintes expressões matemáticas:
   a) $E = m \ X \ c^2$, sendo que E representa a Energia, m representa a Massa e c representa a Velocidade da luz.
   b) $V = X / T$, onde V representa a Velocidade, X representa o Espaço percorrido e T representa o Tempo.
   c) $x' = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$

2. Desenvolva um programa em Delphi para calcular o percentual de aproveitamento de uma disciplina, a partir do cálculo da média semestral ($média = \frac{nota1 + nota2}{2}$).

5 COMANDOS DE DECISÃO

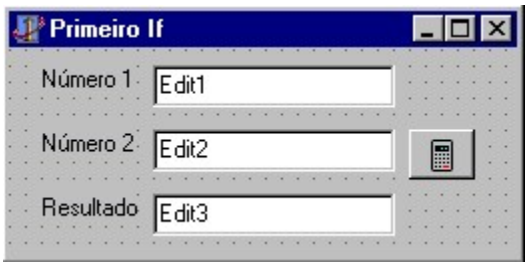
Em qualquer linguagem de programação é necessário que se tomem decisões, ou seja, determinado comando só é executado se alguma condição for verdadeira (ou falsa). Os comandos de decisão podem estar divididos em 3 categorias: decisão simples, decisão composta e múltipla decisão.

Comando de decisão simples

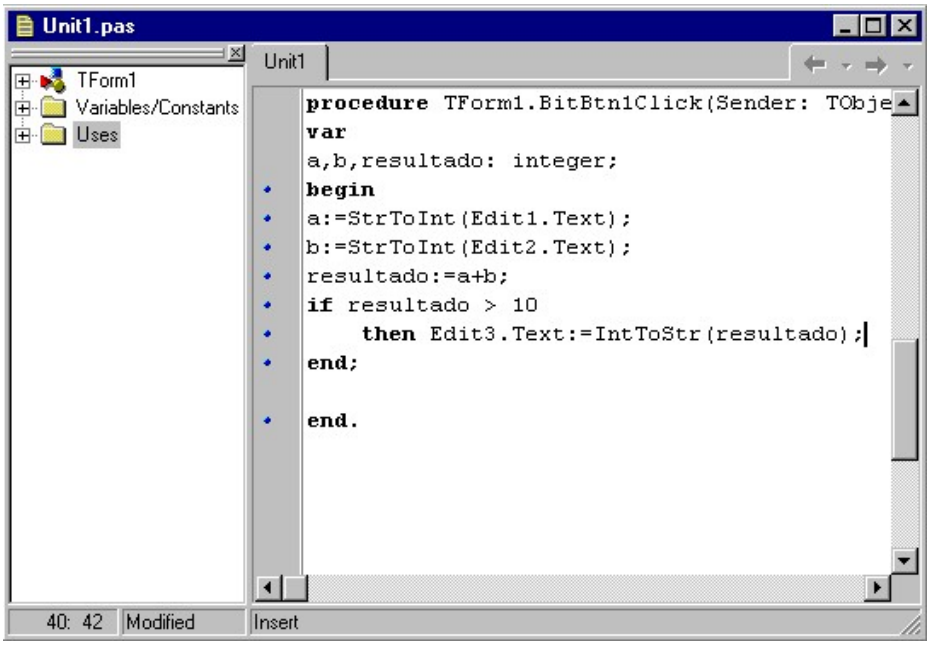
Este comando irá executar uma séria de instruções somente se a condição for verdadeira. Sua sintaxe é:

```
If (condição = verdadeira)
Then Begin
          Instrução1;
          Instrução2;
     End;
```

O primeiro exemplo será construir um programa que mostrará o resultado de uma soma somente se a mesma for maior que 10. A partir do menu **File**, comando **New Application**, introduzir três componentes **Label** e **Edit** da paleta **Standard**. As **label**'s irão indicar "Número 1", "Número 2" e "Resultado". Após isto, acessar a paleta **Additional** e colocar o componente **BitBtn**. Este componente tem a mesma função do componente **Button**, porém pode-se introduzir imagens através da propriedade **Glyph**. As imagens disponíveis no Delphi podem ser carregadas a partir do diretório **\Arquivos de programas\Arquivos comuns\Borland Shared\Images\Buttons**. Selecionar a imagem **calculat**. O **Form** irá assumir a seguinte aparência:



A seguir deve selecionar o componente **BitBtn1** e acessar o evento **OnClick** no **Object Inspector** para digitar o seguinte código:



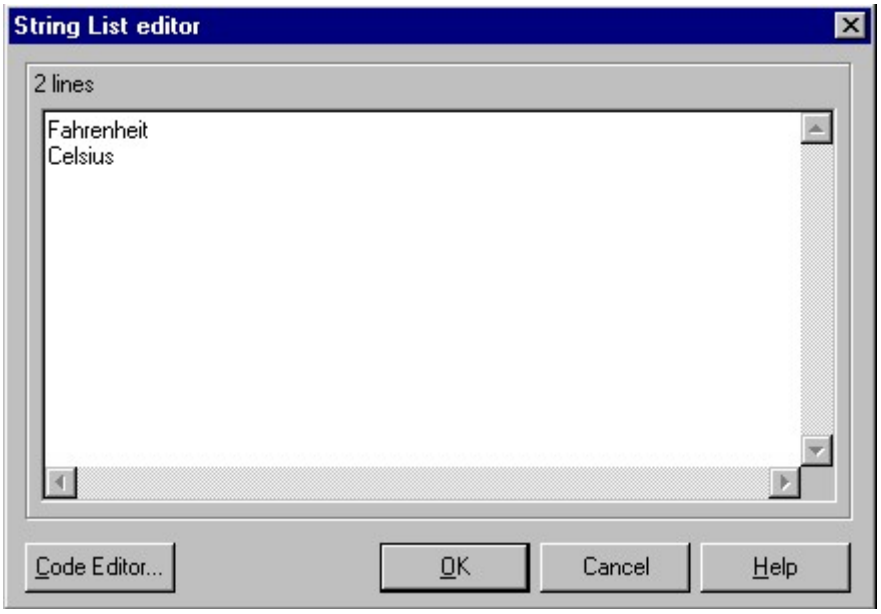
Após isto basta executar o programa e testá-lo para situações cuja soma será maior ou menor que 10.

Comando de decisão composta

Este comando irá executar uma séria de instruções para o caso da condição ser verdadeira e outra série para o caso de ser falsa. Sua sintaxe é:

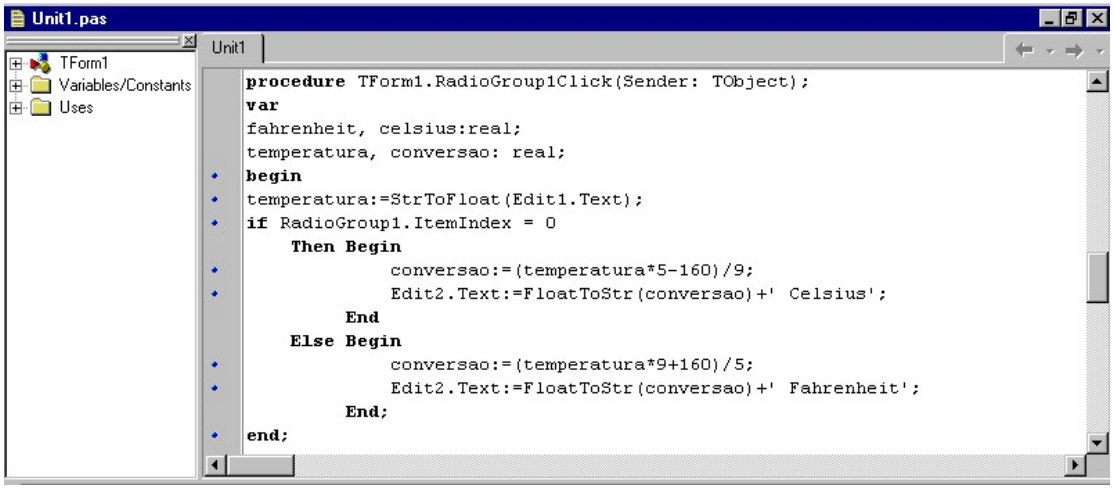
```
If (condição = verdadeira)
Then Instrução1 ← ATENÇÃO
Else Begin
          Instrução2;
          Instrução3;
     End;
```

Para exemplificar este comando, abra uma **New Application** e introduza um componente **Label** cuja propriedade **Caption** será “Informe a temperatura”. Inserir um componente **Edit**. Logo a seguir selecione o componente **RadioGroup**. Este componente permite criar uma caixa onde existirão opções que podem ser selecionadas (é importante frisar que somente uma opção é selecionada de cada vez). A propriedade **Caption** deve ser alterada para “Informe a unidade”. Logo após, dar um duplo clique na propriedade **Itens**, responsável por definir os itens que serão selecionados. Um editor de itens irá se abrir e em cada linha deve ser digitado uma opção. O editor assumirá o seguinte formato:

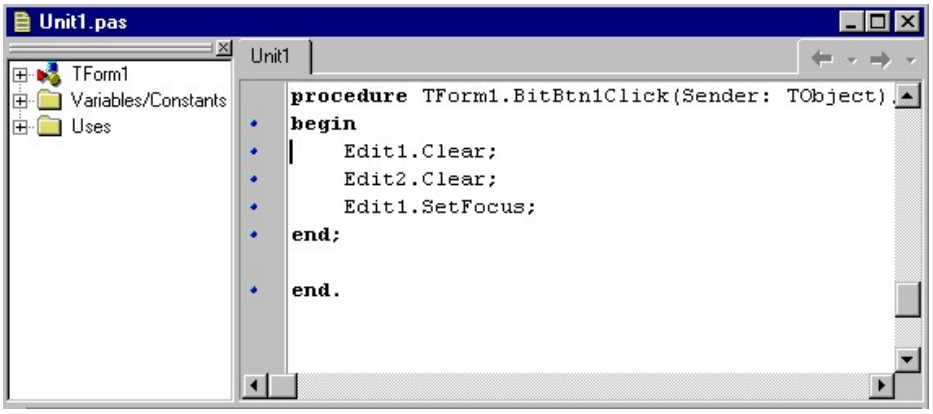


Incluir também uma **Label** e alterar sua propriedade **Caption** para “Temperatura convertida” e um **Edit**. Logo após inserir um componente **BitBtn**, da paleta **Additional** e inserir a imagem **filenew** em sua propriedade **Glyph**. Selecionar o objeto **RadioGroup** e acessar seu evento **OnClick**. O Form irá assumir a seguinte aparência:

Ao digitar as opções no **String list editor** do componente **RadioGroup** atribui-se automaticamente a cada uma destas linhas, um número seqüencial iniciando com 0 (zero), ou seja, a primeira linha terá o índice 0 (zero) e a segunda o índice 1 (um), para diferenciar uma opção de outra. Estes índices são manipulados através da propriedade **ItemIndex** do componente. O seguinte conjunto de instruções deve ser digitado:



Após termos digitado o código para o **RadioGroup**, iremos criar um conjunto de instruções do componente **BitBtn1** para que ao ser clicado, limpe todos os **Edit**'s e faça com que o controle do programa retorne para o **Edit1**, para que seja digitado nova temperatura. Para isto, devemos selecionar o **BitBtn1** e acessar seu evento **OnClick** e digitar os seguintes comandos:



A propriedade **Clear** limpa o conteúdo do **Edit** e a propriedade **SetFocus** posiciona o controle do programa no objeto indicado. Agora basta executar o programa.

Comando de decisão com múltipla escolha

Podem existir situações onde um número elevado de encadeamento de estruturas de decisão tornará o programa difícil de ser entendido. Para evitar esta situação, deve-se usar o comando **Case**, cuja sintaxe é apresentada abaixo:

```
Case variável of
    opção1: comando1;
    opção2: begin
                comando2;
                comando3;
                comando4;
            end;
    opção3: begin
                comando5;
                comando6;
            end;
          else comando7;
End;
```

6 COMANDOS DE REPETIÇÃO

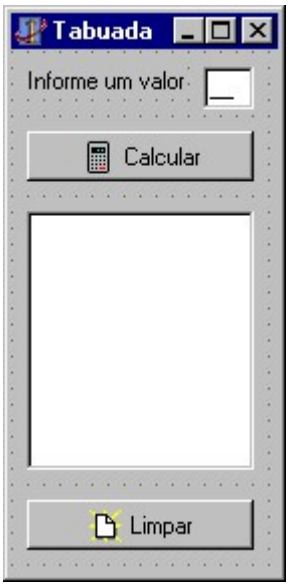
Os comandos de repetição são utilizados para efetuar uma série de operações várias vezes, cuja quantidade é determinada por uma condição.

Comando de repetição While

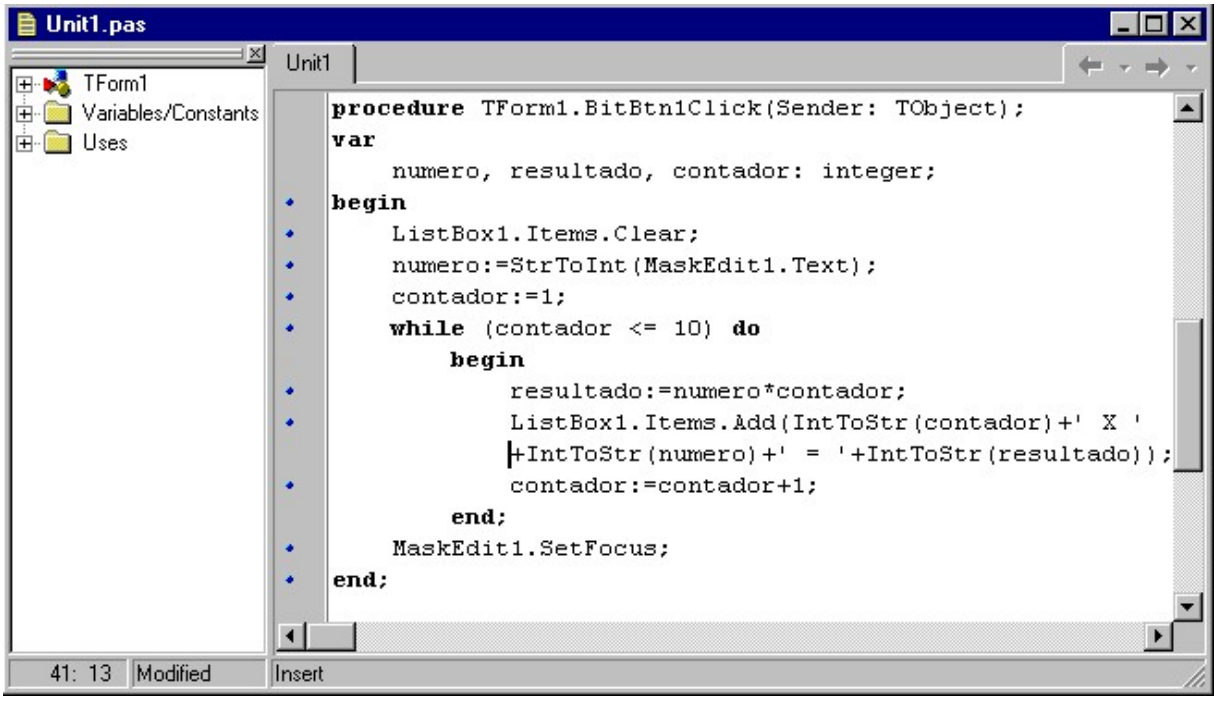
Este comando irá executar uma séria de instruções enquanto a condição ao qual está vinculada for verdadeira. Sua característica é testar a condição antes de efetuar as repetições. Sua sintaxe é:

```
While (condição = verdadeira)
    Then Begin
              Instrução1;
              Instrução2;
         End;
```

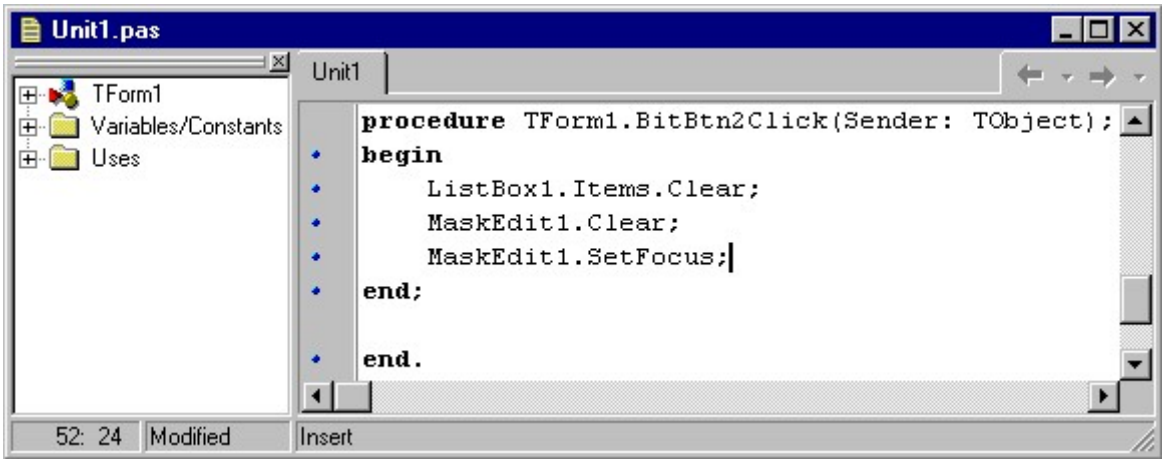
O primeiro exemplo será construir um programa que mostrará o resultado da tabuada (de 1 até 10) de um número digitado. A partir do menu **File**, comando **New Application**, introduzir um componente **Label** da paleta **Standard** e um componente **MaskEdit** da paleta **Additional**. A **Label** irá indicar "Informe um valor" e o **MaskEdit** deverá ser formatado para receber apenas 2 caracteres numéricos. Após isto, na paleta **Standard**, inserir o componente **ListBox**. Este componente tem a função de armazenar várias informações independente da quantidade (se o número de informações for maior que o tamanho definido do componente, automaticamente é criada uma barra de rolagem). Logo em seguida inserir dois componentes **BitBtn** da paleta **Additional**, sendo que um deles será responsável por executar a ação (mostrar o resultado da tabuada) e outro será responsável por limpar os resultados apresentados ou inseridos nos componentes. O **Form** irá assumir a aparência ao lado:



A seguir deve selecionar o componente **BitBtn1** e acessar o evento **OnClick** no **Object Inspector** para digitar o seguinte código:



Para o componente **BitBtn2**, inserir um código para limpar o **ListBox** e o **MaskEdit**, posicionado o foco da aplicação no **MaskEdit**.



Após isto basta executar o programa e testá-lo.

Comando de repetição Repeat

Este comando irá executar uma séria de instruções até que a condição ao qual está vinculado seja falsa. Sua característica é executar o comando e depois testar a condição. Sua sintaxe é:

```
Repeat
    Instrução1;
    Instrução2;
    Instrução3;
Until (condição = verdadeira);
```

Comando de repetição For

As duas formas anteriores de executar uma repetição de comandos estão vinculadas a uma variável (que controla manualmente o número de repetições ou que aguarda uma resposta do usuário). O comando **For** executa repetições automaticamente por um número finito de vezes. Sua sintaxe é apresentada abaixo:

```
For variável := início to fim do
    begin
        comando1;
        comando2;
        comando3;
    end;
```

No exemplo acima, os comandos serão executados até que a <variável> atinja o valor de <fim>, a partir do <início>. Temos, portanto, um incremento desta <variável>. Porém pode-se efetuar um número predeterminado de repetições a partir de um valor que diminui automaticamente a cada execução até atingir um número previamente arbitrado. Nesta situação, o comando **For** assume a seguinte configuração:

```
For variável := fim downto início do
    begin
        comando1;
        comando2;
        comando3;
    end;
```

7 DICAS EM DELPHI

Formatando números reais

Para formatarmos um número como real, definindo o número de casas decimais, basta utilizarmos a função **Str**. Sua sintaxe é:

```
Str (variável_real:5:2, variável_string)
```

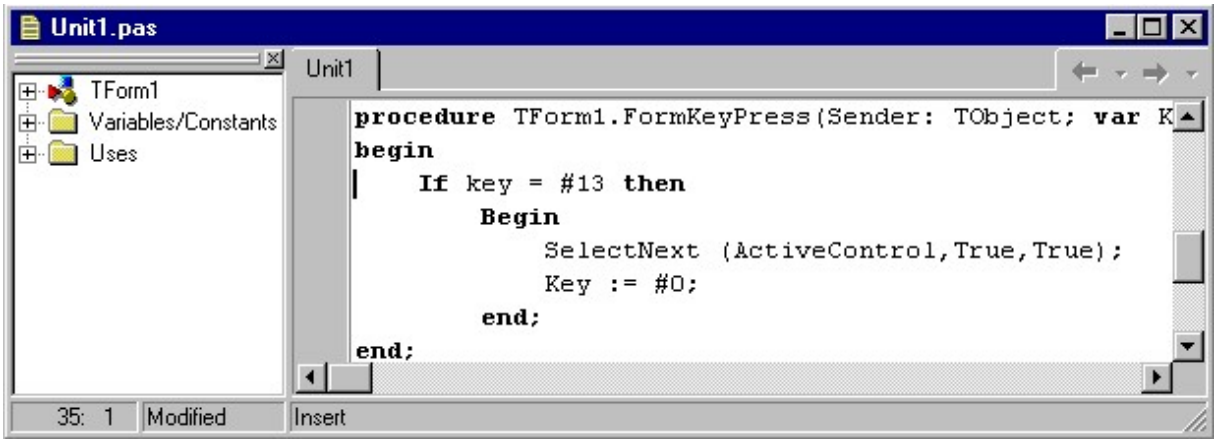
O exemplo pode ser visualizado inserindo-se três componentes **Edit** e **Label** e um **Button** em um formulário. O formulário irá apresentar o formato ao lado. A seguir selecionar o componente **Button1** e acessar o evento **OnClick**. Digitar o seguinte código:


```
procedure TForm1.Button1Click(Sender: TObject);
var
    NumeroOriginal: real;
    NumeroFormatado: string;
begin
    NumeroOriginal:=StrToFloat(Edit1.Text);
    Edit2.Text:=FloatToStr(NumeroOriginal);
    Str(NumeroOriginal:5:2, NumeroFormatado);
    Edit3.Text:=NumeroFormatado;
end;
```

Redefinindo a tecla <Enter>

Sabemos que a movimentação entre os componentes do formulário é possível através da tecla <Tab>, obedecendo a propriedade **TabOrder** de cada componente. Este comportamento pode ser alterado para que a tecla <Enter> assuma esta função. Para exemplificar, iremos criar um formulário com três componentes **Label** e **Edit** e um componente **Button**. O programa terá a função de, ao preencher cada campo e teclar <Enter> passar ao próximo campo. Ao clicar no **Button1**, o programa será encerrado.

Para que isto seja possível devemos clicar no formulário e alterar a propriedade **KeyPreview** para **True**. Logo em seguida selecionar o evento **OnKeyPress** do **Form1** e digitar o seguinte código:



Para finalizar o programa basta inserir o comando **Close** no evento **OnClick** do componente **Button1**.

Exibindo caixas de mensagens

Podemos utilizar caixas de mensagens para obter uma maior interatividade entre o sistema e o usuário, permitindo a exibição de mensagens de informação, erro, interrogação e assim sucessivamente. Sua sintaxe é:

MessageDlg (mensagem, tipo, [botões], ContextoAjuda)

Onde:
Mensagem: especifica qual a mensagem será exibida ao usuário.
Tipo: determina o ícone e o tipo da mensagem exibida ao usuário. Podem ser:

- mtWarning: advertência;
- mtError: erro;
- mtInformation: informação;
- mtConfirm: conformação;
- mtCustom: apresenta o nome do projeto.

Botões: determina os botões que serão exibidos dentro da caixa de mensagem.

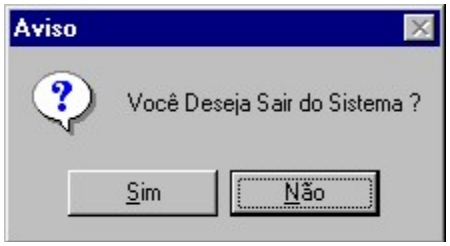
- mbYes: sim;
- mbNo: não;
- mbOk: ok;
- mbCancel: cancelar;
- mbHelp: ajuda;
- mbAbort: abortar;
- mbRetry: repetir;
- mbIgnore: ignorar;
- mbAll: todos os botões.

ContextoAjuda: determina a tela que será exibida quando pressionarmos e tecla F1, somente se criarmos o Help on-line. Caso não se tenha criado o Help on-line, seu valor será 0 (zero).

Para exemplificarmos basta acrescentarmos a linha `MessageDlg('O programa será encerrado',mtInformation,[mbOk],0);` no evento **OnClick** do componente **Button1**, utilizado no exemplo anterior.

Uma alternativa para a função **MessageDlg** é o teste para a função **MessageBox**. Se digitarmos o código abaixo no evento **OnClick** do componente **Button1**, utilizado no exemplo anterior, teremos o seguinte resultado:

```
begin
    If MessageBox(0,'Você Deseja Sair do Sistema ?', 'Aviso',
    Mb_IconQuestion + Mb_YesNo + Mb_DefButton2 + mb_TaskModal)
    = IdYes
        then Close;
end;
```



Trabalhando com mais de um formulário

Existirão situações onde é necessário mais de um formulário. Para ilustrar esta situação, vamos inserir dois componentes **Button** no **Form1**. Cada **Button** irá chamar um novo **Form**. Para isto, vamos alterar a propriedade **Caption** de cada **Button** para Formulário1 e Formulário2.

Após isto, digitar no evento **OnClick** do componente **Button1**, o seguinte código:

```
Form1.Hide; // Esconde o Form1
Form2.Show; // Mostra o Form2
```

No evento **OnClick** do componente **Button2**, digitar o seguinte código:

```
Form1.Hide; // Esconde o Form1
Form3.Show; // Mostra o Form3
```

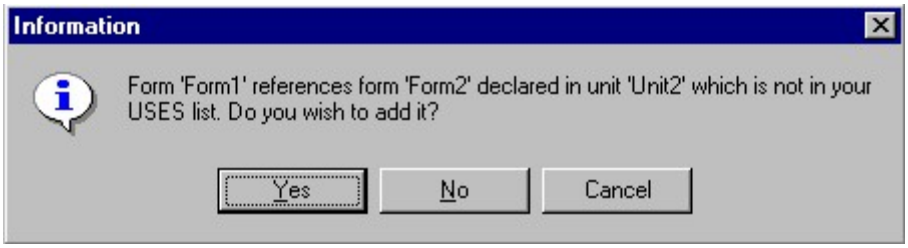
Digitado as linhas de código referentes a cada **Button**, devemos inserir dois novos **Form**'s, acessando o menu **File**, opção **New Form**. Em cada novo **Form** devemos inserir um componente **Button** e digitar para o **Button1** do **Form2**:

```
Form2.Hide; // Esconde o Form2
Form1.Show; // Mostra o Form1
```

Para o **Button1** do **Form3**, digitar:

```
Form3.Hide; // Esconde o Form3
Form1.Show; // Mostra o Form1
```

Ao compilar o programa, será apresentada a seguinte caixa de mensagem:



A tradução literal da caixa é “Formulário ‘Form1’ faz referência ao formulário ‘Form2’ declarado na unit ‘Unit2’ que não está em sua lista de Uses. Você gostaria de adicioná-lo?”. Isto significa que a **Unit2** que controla o **Form2** não está sendo reconhecida pela **Unit1** (que controla o **Form1**). Portanto devemos clicar no botão **Yes** para fazer as devidas referências.

Existem alternativas para os métodos **Hide** e **Show**: no **Button1** do **Form1** podemos simplesmente digitar a linha `Form2.ShowModal` e no **Button1** do **Form2**, o comando `Close`.

Criando menus

Para trabalharmos com menus basta inserirmos o componente **MainMenu** no **Form1**. Ao clicarmos na propriedade **Items** teremos acesso ao editor de menus deste componente. Na propriedade **Caption** do editor de menus inserimos o texto que irá corresponder a cada menu. Colocando o caracter <&> antes de uma letra do texto, poderemos acessar aquele menu com a combinação da tecla <Alt> mais a letra. Para associar uma ação a cada menu, basta clicar sobre o mesmo e inserir os códigos necessários.

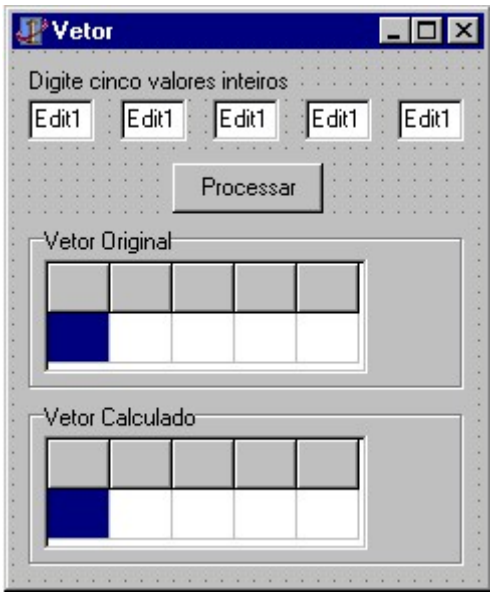
8 ESTRUTURAS MATRICIAIS

A função de estrutura matricial é agrupar várias informações de mesma natureza em uma variável de memória. Para isto existe a necessidade de indexar esta variável, identificando desta forma, a posição onde a informação está localizada dentro da variável. As estruturas matriciais podem ser classificadas com vetores (quando possuem apenas 1 dimensão) ou matrizes (quando possuem mais de 1 dimensão).

Vetores

Para ilustrarmos um exemplo com vetores vamos desenvolver um programa que leia 5 números inteiros e os armazene em um vetor. Logo em seguida crie um novo vetor acompanhando a seguinte regra: se o valor do índice for par, o valor do elemento deverá ser multiplicado por 5; se for ímpar deverá ser somado a 5.

A partir da paleta de componentes **Standard**, introduzir 5 componentes **Edit**, 3 **Label**, 2 **GroupBox** e 1 **Button**. Da paleta **Additional**, inserir 2 componentes **StringGrid**. O formulário assumirá a aparência ao lado. No evento **OnClick** do **Button1** iremos definir o vetor original e o calculado, declarando antes da seção **Implementation** os vetores:




```
var
    VetorOriginal: array [1..5] of integer;
    VetorCalculado: array [1..5] of integer;
```

Estas variáveis irão agir como Públicas, visualizadas por todos os eventos necessários no **Form1**.

Na procedure **Tform1.Button1Click** inserir uma variável para controlar o índice dos vetores. **Lembrar que o índice sempre deve ser declarado como uma variável privada!!!** Logo após digitar o código a seguir:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    Indice: integer;
begin
    VetorOriginal[1]:= StrToInt(Edit1.Text);
    VetorOriginal[2]:= StrToInt(Edit2.Text);
    VetorOriginal[3]:= StrToInt(Edit3.Text);
    VetorOriginal[4]:= StrToInt(Edit4.Text);
    VetorOriginal[5]:= StrToInt(Edit5.Text);

    with StringGrid1 do
        for Indice:=1 to 5 do
            begin
                cells[Indice-1,0]:=IntToStr(Indice);
                cells[Indice-1,1]:=IntToStr(VetorOriginal[Indice]);
            end;

    with StringGrid2 do
        for Indice:=1 to 5 do
```

```
            begin
                cells[Indice-1,0]:=IntToStr(Indice);
                if Indice mod 2 = 0
                    then cells[Indice-1,1]:=IntToStr(VetorOriginal[Indice]*5)
                    else cells[Indice-1,1]:=IntToStr(VetorOriginal[Indice]+5);
            end;
end;
```

Para ilustrar a aplicação de vetores iremos fazer um programa para realizar a ordenação de 10 nomes, inserindo em um novo formulário 1componente **Label**, 10 componentes **Edit**, 1 **Button** e 1 **ListBox**. A seguir, digitar no evento **OnClick** do **Button1**, o código abaixo:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    nome: array [1..10] of string[20];
    i, j: byte;
    auxiliar: string[20];
begin
    ListBox1.Items.Clear;
    nome[01]:= Edit1.Text;
    nome[02]:= Edit2.Text;
    nome[03]:= Edit3.Text;
    nome[04]:= Edit4.Text;
    nome[05]:= Edit5.Text;
    nome[06]:= Edit6.Text;
    nome[07]:= Edit7.Text;
    nome[08]:= Edit8.Text;
    nome[09]:= Edit9.Text;
    nome[10]:=Edit10.Text;

    For i:= 1 to 9 do
        For j:= i+1 to 10 do
            If (nome[i] > nome[j])
                then begin
                        auxiliar:= nome[i];
                        nome[i]:= nome[j];
                        nome[j]:= auxiliar;
                     end;

    For i:= 1 to 10 do
        ListBox1.Items.Add(nome[i]);
end;
```

Matrizes

Matrizes são estruturas com mais de uma dimensão, onde é possível controlar os conteúdos presentes em linhas e colunas. Como exemplo, iremos demonstrar a criação de uma matriz a partir de elementos digitados diretamente nela. Primeiramente iremos abrir um novo formulário e inserir um componente **GroupBox** com as propriedades **Caption** (Preencha a matriz), **Height** (153), **Left** (8), **Top** (16), **Width** (249) alteradas. Depois devemos implementar os seguintes passos:

1º. Na seção **type**, inserir a linha **procedure** `CriaMatriz(Sender: Tobject);` logo após a linha `GroupBox1: TGroupBox`;
2º. Na seção **var**, inserir a linha `matriz: array [1..4, 1..5] of TEdit;` logo após a linha `Form1: TForm1`;
3º. Logo após a diretiva ***{$R *.DFM}***, digitar o seguinte código:

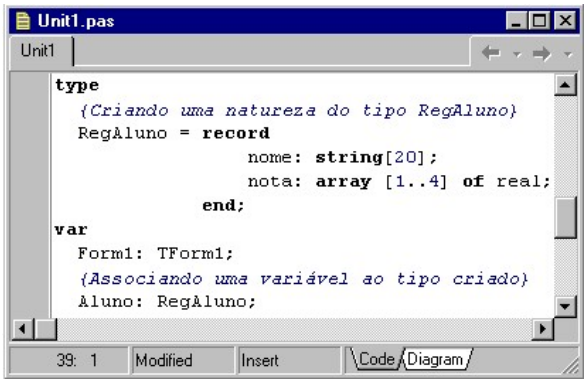
```
procedure TForm1.CriaMatriz(Sender: TObject);
var
    retorno: string;
    linha, coluna: integer;
    i, j, x: byte;
begin
    x:=1;
    linha:=40;
    for i:=1 to 4 do
        begin
            coluna:=16;
            for j:=1 to 5 do
                begin
                    matriz[i,j]:=TEdit.Create(self);
                    matriz[i,j].parent:=self;
                    matriz[i,j].left:=coluna;
                    matriz[i,j].top:=linha;
                    matriz[i,j].width:=41;
                    matriz[i,j].height:=21;
                    str(x,retorno);
                    matriz[i,j].name:='Edit'+retorno;
                    matriz[i,j].text:='';
                    coluna:=coluna+48;
                    x:=x+1;
                end;
            linha:=linha+32;
            matriz[1,1].setfocus;
        end;
end;
```

4º. Finalmente associar a procedure CriaMatriz ao evento **OnActivate** do **Form1**.

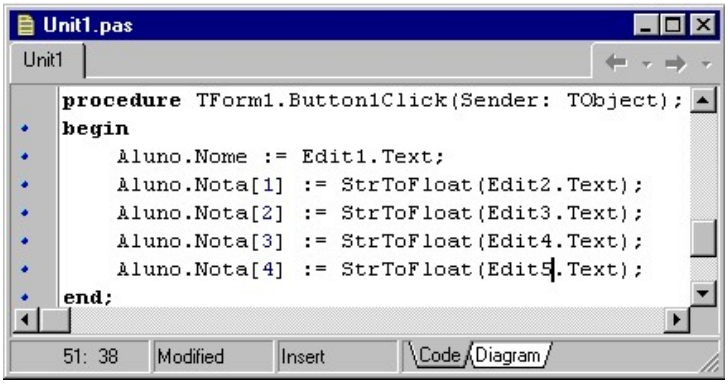
Registros

Com vetores e matrizes percebemos que é possível armazenar vários valores ao mesmo tempo, porém, com apenas um tipo de dado (integer, real, etc). Para contornar esta situação pode-se trabalhar com registros. Vamos imaginar uma situação onde é necessário guardar uma string e quatro reais, representado pelos dados contidos no formulário ao lado.

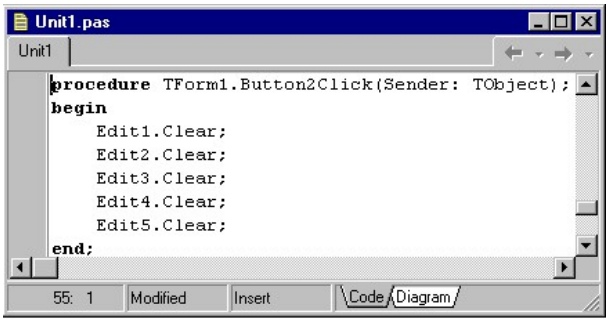
O primeiro passo é definir na seção **type** da **Unit1** uma natureza de variável que permita armazenar strings e reais e em seguida criar uma variável que receba dados da forma recém definida.

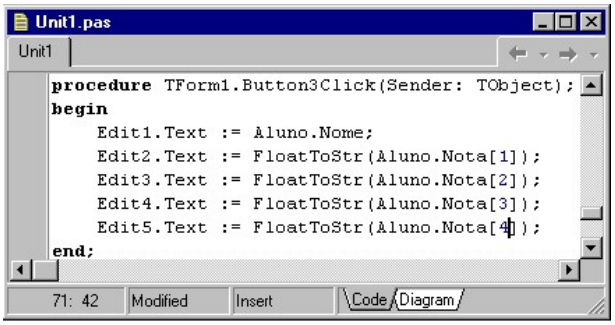


A seguir basta implementar, nos eventos **OnClick** de cada botão, as ações a serem executadas quando clicados. No **Button1**, iremos digitar as seguintes instruções para armazenar os valores dos **Edit**s no registro (RegAluno):



Os Button2 e Button3 terão as seguintes instruções respectivamente:





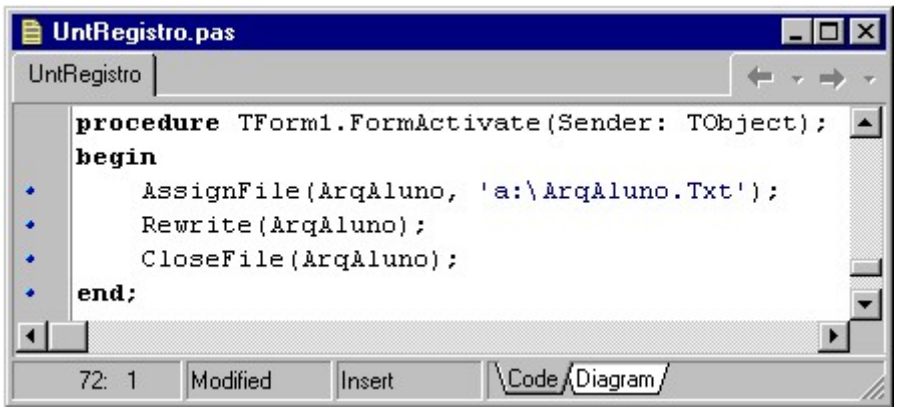
9 ARMAZENAMENTO DE INFORMAÇÕES

Arquivos

É possível armazenar as informações em arquivos de forma a recupera-las sempre que necessário. Para isto vamos criar um novo formulário com as mesmas características do exemplo da subseção 9.3, apenas mudando a propriedade **Caption** dos **Button**s para Cadastra, Pesquisa e Exclui. Em seguida associar o arquivo à estrutura do registro na **Unit1**.

```
var
    Form1: TForm1;
    Aluno: RegAluno;
    {Associando o arquivo à estrutura do registro}
    ArqAluno: File of RegAluno;
```

A seguir deve-se abrir o arquivo quando o formulário for ativado. Portanto no evento **OnActivate** do **Form1** deve-se digitar as instruções abaixo:



A instrução **AssignFile** associa o nome lógico do arquivo (declarado na seção **var**) ao nome físico. A instrução **Rewrite** cria o arquivo e o mantém aberto. Finalmente **CloseFile** fecha o arquivo.

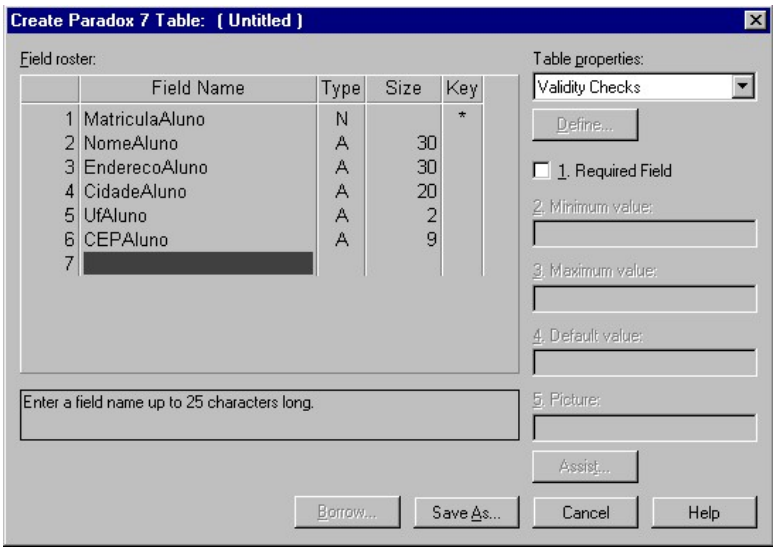
Para cadastrar efetivamente os dados, devemos associar o registro ao arquivo, digitando as seguintes instruções no evento **OnClick** do **Button1**:

Tabelas

As informações podem ser armazenadas utilizando-se tabelas de bancos de dados. Tabelas são um conjunto de registros agrupados conforme a necessidade de armazenamento e recuperação das informações.

Criação de tabelas

1º) Menu **Tools**, opção **Database Desktop**;
2º) Menu **File**, opção **New/Table**;
3º) Pode-se escolher vários tipos de bancos de dados. Iremos trabalhar com o **Paradox 7**;
4º) Informar o Nome do campo (**Field Name**), o tipo (**Type**), seu tamanho (**Size**) e se o mesmo é campo chave (**Key**);
5º) Após definir todos os parâmetros, salvar a tabela através do botão **Save As**.



Componentes de banco de dados

Uma vez criada a tabela é necessário confeccionar um formulário para a manipulação desta tabela. Para determinarmos a ligação entre o formulário e a tabela, utilizaremos o componente **Table**. Inserir o componente e atualizar as seguintes propriedades:

DatabaseName: determina a pasta onde a tabela será manipulada. Determinar o caminho desta tabela.
IndexFieldsNames: determina qual campo será utilizado como índice da tabela.
Name: nome do componente **Table**.
TableName: especifica qual tabela será utilizada.
Active: determina se a tabela está aberta (**True**) ou fechada (**False**). Alterar para **True**.

A seguir devemos inserir o componente **DataSource** da paleta **Data Access**, que será responsável por "ligar" o componente **Table** com os demais componentes de controle de dados. Devemos alterar as seguintes propriedades:

DataSet: determina a origem dos dados (nome do componente **Table**).
Name: nome do componente **DataSource**.

Confeccionando o formulário

O próximo passo para a criação de um programa que utiliza tabelas é a confecção do formulário a partir de um duplo clique no componente **Table**. Irá mostrar o **Form1.Table**, onde devemos clicar com o botão direito sobre o editor de campos e selecionar a opção **Add all fields** para selecionar todos os campos da tabela que serão utilizados no formulário. Para inseri-los basta seleciona-los e arrasta-los para o formulário.

Manipulando os registros

O componente **DBNavigator**, da paleta **Data Controls** permite a navegação, edição, deleção de registros da tabela. Para que possamos associa-lo à tabela devemos indicar o **DataSource** responsável pela tabela, na propriedade **DataSource**.

A manipulação de registros pode ocorrer através de botões customizados. Vamos inserir dez **BitBtn** e inserir no evento **OnClick** de cada um deles, na ordem, o seguinte código:

BitBtn1: Table1.First;

BitBtn2: Table1.Prior;

BitBtn3: Table1.Next;

BitBtn4: Table1.Last;

BitBtn5: Table1.Insert;

BitBtn6: Table1.Post;

BitBtn7: Table1.Edit;

BitBtn8: Table1.Delete;

BitBtn9: Table1.Cancel;

BitBtn10: Table1.Close;

10 BIBLIOGRAFIA

LITE. **Borland Delphi 5 – Passo a Passo.** São Paulo: Makron Books, 2000.

MANZANO, José Augusto; MENDES, Sandro Santa Vicca. **Estudo dirigido de Delphi 5**. São Paulo: Érica, 1999.

TEIXEIRA, Silvio; PACHECO, Xavier. **Delphi 5 – guia do desenvolvedor**. Rio de Janeiro: Campus, 2000.